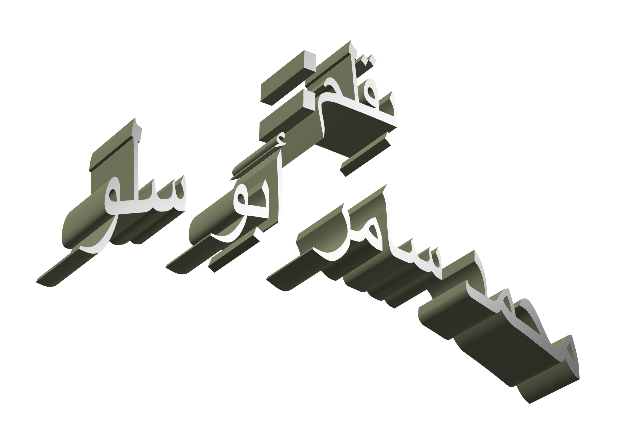

معالجة الأخطاء بالفيجول بايزيك دوت نيت


تقديم
محمد سامر أبو سلو

ويضم المواضيع التالية:

معالجة الأخطاء
تنقيح الأخطاء في برنامجك
الاستثناءات Exceptions اصطياد الأخطاء ومعالجتها

معالجة الأخطاء

متى نستخدم معالجات الأخطاء

يمكنك استخدام معالجات الأخطاء في أي وضع يولد احتمال لحدوث خطأ يوقف تنفيذ البرنامج سواء كان ذلك الخطأ متوقعا أم غير متوقع. وبشكل عام تستخدم معالجات الأخطاء لإدارة أحداث خارجية قد تؤثر على مسار تنفيذ البرنامج كفشل في الوصول للشبكة أو قرص مضغوط غير موجود أو طابعة أو ماسح ضوئي غير مشغلين مثلا ومن هذه المشاكل المحتملة التي تحتاج إلى معالجات أخطاء:

- **شبكة/انترنت** مشكلة في المخدمات أو تجهيزات الاتصال
- **قواعد بيانات** عدم القدرة على إنشاء اتصال أو تنفيذ استعلام أو أن قاعدة البيانات تعيد خطأ ما
- **سواقات الأقراص** قرص غير مهيأ أو مهيأ بصورة غير صحيحة أو قرص قابل للإزالة غير مدخل بشكل جيد
- أو قطاعات تالفة أو حتى قرص ملئ
- **مشاكل المسارات** مسار خاطئ أو غير صحيح
- **مشاكل الطابعة** طابعة مطفأة أو بدون ورق أو غير متوفرة لسبب ما
- **مشاكل برمجيات** مكونات أو مكتبات يعتمد عليها ملف للتنفيذ ناقصة أو غير منصبة بصورة صحيحة أو
- وجود تعارض أو عدم توافقية بين بعض المكتبات
- **مشاكل أمان** البرنامج يحاول القيام بعملية غير مسموحة أو أن المستخدم الذي يشغل البرنامج لا يمتلك
- الصلاحيات الكافية لإتمام تنفيذ العملية
- **مشاكل ذاكرة** مصادر نظام غير كافية
- **مشاكل الحافظة** مشاكل في تبادل البيانات مع حافظة ويندوز
- **مشاكل منطقية** مشاكل صيغة أو مشاكل منطقية لم يستطع المترجم كشفها

كتلة Try ... Catch

كتلة الكود التي تستخدم لمعالجة أخطاء زمن التنفيذ تدعى Try ... Catch حيث يمكنك كتابة عبارة Try ضمن إجراء معالجة الحدث قبل الكود الذي تتوقع أن يولد مشكلة وتليها مباشرة عبارة Catch فإن حدث خطأ في زمن التنفيذ فيتم تنفيذ مجموعة من العبارات ضمن كتلة Catch ثم ستنفذ مجموعة من العبارات الاختيارية في كتلة Finally كما يمكن في بعض الحالات تعشيش عدة بلوكات Try ... Catch داخل بعضها و تكون الصيغة العامة لكتلة Try ... Catch

```
Try
        Statements that might produce a run-time error
        العبارات الممكن أن تولد خطأ
Catch
        Statements to run if a run-time error occurs
        العبارات التي تنفذ في حالة حدوث خطأ
Finally
        Optional statements to run whether an error occurs or not
        عبارات اختيارية ستنفذ إن حدث خطأ أم لا
End Try
```

حيث تشكل عبارة Try بداية تعريف معالجة الخطأ في حين أن العبارات Try و Catch و End Try هي عبارات إجبارية والعبارة Finally اختيارية. ويدعى الكود المتواجد بين عبارتي Try و Catch بالكود المحمي بسبب أن أخطاء زمن التنفيذ المتولدة ضمن ذلك الكود لا تتسبب في توقف البرنامج عن العمل

فمثلا إن حاولنا فتح ملف صورة وتحميله في صندوق الصورة يمكننا وضع ذلك الكود ضمن كتلة Try ... Catch وذلك لحماية البرنامج من الأخطاء التي قد تحدث في زمن التنفيذ

```
Try
        PictureBox1.Image = _
                System.Drawing.Bitmap.FromFile("D:\FileOpen.bmp")
Catch
```

```
        MsgBox("Please insert the disk in drive D")
End Try
```

والمثال التالي يبين لنا كيفية تعشيش كتل Try ... Catch

```
Try
        PictureBox1.Image = _
                System.Drawing.Bitmap.FromFile("D:\FileOpen.bmp")
Catch
        MsgBox("Please insert the disk in drive D, Then Click Ok")
        Try
                PictureBox1.Image = _
                        System.Drawing.Bitmap.FromFile("D:\FileOpen.bmp")
        Catch
                MsgBox("File Load feature disabled")
        End Try
End Try
```

الغرض Err

من الأغراض الموروثة المفيدة الباقية من نسخ فيجول بايزيك السابقة الغرض Err بنسخته المحدثة بمعلومات مفصلة لمعالجة الأخطاء لكل خطأ زمن تنفيذ يحدث في البرنامج ومع ذلك هناك طرق أحدث لإدارة الأخطاء في الفريموورك مثل الغرض Exception القوي وتشكل الخصائص Err.Number و Err.Description الخصائص الأكثر إفادة لتحديد أخطاء زمن التنفيذ فالخاصية Err.Number تحتوي على رقم الخطأ الخاص بآخر خطأ زمن تنفيذ حدث مؤخرا والخاصية Err.Description تحتوي على رسالة قصيرة تطابق رقم الخطأ الذي حدث مؤخرا وتلك الخاصيتان تمكناك من التعرف على الأخطاء التي حدثت مؤخرا وتحديد الاستجابة المناسبة لها وقد تعطي المستخدم رسالة عن كيف يمكن أن يتصرف في حالة حدوث خطأ معين كما يمكنك تفريغ الخطأ بواسطة الطريقة Err.Clear ولكن إن استخدمت الغرض Err داخل كتلة Catch فيصبح من غير الضروري تفريغ الخطأ لأنه لا يتم الدخول إلى كتلة Catch إلا إن حدث خطأ والمثال التالي يتعرف على عدة أخطاء زمن التنفيذ باستخدام الغرض Err

```
Try
        PictureBox1.Image = _
                System.Drawing.Bitmap.FromFile("D:\FileOpen.bmp")
Catch When Err.Number = 53   'If File Not Found Error
        MsgBox("Check pathname and disk drive")
Catch When Err.Number = 7   'If File Out Of Memory Error
        MsgBox("Is this really a bitmap", , Err.Description)
Catch
        MsgBox("Problem loading file", , Err.Description)
End Try
```

العبارة Exit Try

وهي تستخدم للخروج من كتلة Try ... Catch بشكل مشابه للعبارات Exit For و Exit Sub المألوفة حيث باستخدامها تخرج كليا من كتلة Try ولكن إن كان قسم Finally موجودا فسيتم تنفيذه ولكن عبارة Exit Try تجعلك تقفز فوق بقية عبارات Try ... Catch الباقية

```
Try
        If PictureBox1.Enabled = False Then Exit Try
        PictureBox1.Image = _
                System.Drawing.Bitmap.FromFile("D:\fileopen.bmp")
Catch
        Retries +=1
        If Retries <= 2 Then
                MsgBox("Please insert the disc in drive D")
        Else
```

```
            MsgBox("File Load feature disabled")
            Button1.Enabled = False
        End If
End Try
```

مقارنة معالجات الأخطاء مع التقنيات الدفاعية للبرمجة

استخدام معالجات الأخطاء ليست الطريقة الوحيدة لحماية برنامجك من حدوث أخطاء زمن التنفيذ فقطعة الكود التالية تستخدم الطريقة File.Exists في مجال الأسماء System.IO في مكتبة فئات الفريموورك للتأكد من أن الملف موجود فعلا قبل محاولة فتحه

```
If file.Exists("D:\fileopen.bmp") Then

    PictureBox1.Image = _

        System.Drawing.Bitmap.FromFile("D:\fileopen.bmp")

Else

    MsgBox("Cannot find fileopen.bmp on drive D.")

End If
```

في الكود السابق لا تعتبر عبارة If معالج خطأ حقيقي لأنها لا تستطيع منع خطأ زمن التنفيذ من إيقاف تنفيذ البرنامج وبدلا عن ذلك فطريقة التحقق هذه التي يستخدمها بعض المبرمجين تدعى بالبرمجة الدفاعية. فهي تستخدم وظيفة مفيدة في مكتبة فئات الفريموورك للتأكد من العملية التي ستجري على الملف قبل محاولة فتحه الفعلية. وفي هذه الحالة خاصة فالتأكد من وجود الملف باستخدام الطريقة File.Exists هي أسرع من انتظار فيجول بايزيك لإطلاق الاستثناء واستعادته من خطأ زمن التنفيذ باستخدام معالجات الأخطاء.
وهنا يظهر لدينا سؤال: متى يجب علينا استخدام طريقة البرمجة الدفاعية ومتى يجب علينا استخدام معالجات الأخطاء؟ ويكون الجواب هو أنه يجب عليك استخدام مزيج من الطريقتين في كودك حيث تكون طريقة البرمجة الدفاعية هي الأكثر فعالية لمعالجة المشاكل المحتملة. وكما ذكرنا سابقا فالطريقة File.Exists أسرع من بلوك Try ... Catch لمعالجة الأخطاء لذا فمن المنطقي استخدام تقنية البرمجة الدفاعية من أجل الأخطاء التي تتوقع حدوثها بشكل متكرر واستخدام التراكيب الخاصة بمعالجة الأخطاء إذا كان لديك أكثر من شرط لفحصه وتريد تزويد مستخدم برنامجك بعدد من الخيارات كاستجابة لذلك الخطأ كما تمكنك من معالجة الأخطاء التي قد لا تتوقعها.

تنقيح الأخطاء في برنامجك، Debugging Your Application

عند تطوير تطبيق ما يواجه المبرمج مشاكل وأخطاء تظهر أثناء التنفيذ أو الترجمة وتنقسم هذه الأخطاء إلى عدة أنواع: خطأ بالصيغة وهذا يسهل اكتشافه حيث لن يقوم الـ Compiler بترجمة المشروع وتنفيذه إن وجد خطأ من هذا النوع وقد تعترض عليه بيئة التطوير أثناء كتابتك لشفرة البرنامج - خط أحمر تحت العبارة - وأخطاء وقت التنفيذ وهذه أخطاء طارئة تحدث أثناء تنفيذ البرنامج ويجب مراقبتها في الشيفرة وهنا نستخدم عبارة Try ... Catch لحصر تلك الأخطاء وتجاوزها مثل عندما يحاول البرنامج فتح ملف قد يكون غير متوفر للفتح لأسباب متعددة مرتبطة ببيئة التشغيل ونوع آخر وهي أخطاء منطقية في الكود حيث تلاحظ أن صيغة الأوامر صحيحة ولكن البرنامج لا يقوم بالعمل كما يجب ففي هذه الحالة طرق تجاوز النوعين السابقين من المشاكل لن تفيدك وستضطر لاستخدام أدوات التنقيح Debugging tools لحصر وتصحيح تلك المشاكل وفيما يلي بعض النقاط التي تساعدك على استخدام هذه الأدوات لتجاوز المشاكل من النوع الأخير

يمكنك وضع نقاط التوقف Break Points لإيقاف تنفيذ البرنامج عند سطر معين ويمكن بعد التوقف متابعة تنفيذ البرنامج باستخدام F11 للمتابعة سطر سطر أو F5 لمتابعة تنفيذ البرنامج حيث يمكن وضع نقاط التوقف أو إزالتها باختيار البند Toggle Breakpoints من قائمة Debug أو ضغط المفتاح F9 أو النقر على الهامش الرمادي بجانب السطر المراد التوقف عنده وتظهر دائرة حمراء بجانب السطر دلالة على وضع نقطة التوقف عنده

لتشغيل البرنامج مع التنقيح اختر Start Debugging من قائمة Debug أو اضغط F5 ولتشغيله بدون تنقيح اختر Start Without Debugging أو اضغط Ctrl+F5

يمكنك ضغط المفتاح F11 لبدء البرنامج مع التنقيح سطر سطر

اضغط F11 سترى أنك قد انتقلت لأول سطر كود سيتم تنفيذه ولمتابعة تنفيذ البرنامج سطر سطر تابع ضغط F11 سترى في كل مرة أنه قد نفذ سطرا آخر من البرنامج حيث يمكنك استخدام هذه الطريقة للفهم الدقيق لكيفية تنفيذ البرنامج كما أن F10 تقوم بنفس عمل F11 تقريبا إلا أنها إذا واجهت إجراء ضمن الكود الذي يتم تنقيحه فإنها تمرر التنفيذ للسطر التالي فورا دون المرور بتنفيذ ذلك الإجراء سطر سطر كما تفعل F11 التي تنتقل لذلك الإجراء وتنفذه سطر سطر قبل العودة لتنفيذ باقي الكود المستدعي للإجراء

يمكنك إيقاف تنفيذ البرنامج وذلك إما بالضغط على زر التوقف من شريط الأدوات أو Shift-F5

اضغط F5 لتشغيل البرنامج وبهذا يبدأ تشغيل المنقح ويستمر تنفيذ الكود حتى يمر على نقطة توقف Break Point وعندها يتوقف عند السطر المحدد بنقطة التوقف المحددة سابقا وبينما أنت في وضع التوقف يمكنك متابعة بيانات الفئات في البرنامج عبر نافذتي Autos و Locals

نافذة locals تريك جميع المتغيرات المعرفة ضمن مجال التنفيذ الحالي حيث يمكنك استخدامها لرؤية جميع خصائص تلك المتغيرات وقيمها ونافذة Autos تعمل بطريقة مشابهة ولكنها ترينا متغيرات قد لا تكون معرفة ضمن مجال التنفيذ الحالي

إذا أوقفت مؤشر الفأرة فوق متغير أو خاصية ما وأنت في وضع التوقف ستلاحظ ظهور نافذة صغيرة تظهر لك تلك الخاصية وقيمتها ويمكنك عند الحاجة تغيير تلك الخاصية بالنقر المزدوج عليها وكتابة قيمة جديدة أو الضغط بزر الفأرة اليميني عليها ثم اختيار Edit Value من القائمة وتغيير تلك القيمة حيث يمكنك بعدها متابعة التنقيح باستخدام F11

لتغيير السطر التالي الذي سيتم تنفيذ الكود عنده فقط انقر بزر الفأرة اليميني على الخاصية واختر من القائمة Set Next Statement ستلاحظ تغير مكان السهم الأصفر الذي يدل على السطر التالي الذي سيتم تنفيذه

عندما توقف مؤشر الفأرة في وضع التوقف فوق نوع بيانات مركب مثل Me التي تشير للفئة الحالية مثلا أو متغير يشير إلى فئة أو تركيب ما أو قد يشير إلى Dataset مثلا يمكنك بالضغط على إشارات + لتنقل ورؤية جميع خصائص تلك الفئة أو نوع البيانات المركب أو تغييرها وذلك بنفس الطريقة التي تستخدمها للتنقل بين عناصر TreeView

إذا أردت تنفيذ البرنامج حتى يصل لسطر معين يمكنك فعل ذلك مباشرة بدون الضغط على F11 للتنفيذ وذلك بالضغط بزر الفأرة اليميني على ذلك السطر واختيار Run to Cursor حيث سيتم تنفيذ البرنامج حتى ذلك السطر

لمراقبة قيمة متغير بشكل مستمر نستخدم Watch window حيث يمكنك النقر بزر الفأرة اليميني على ذلك المتغير واختيار Add Watch حيث يمكنك رؤية ذلك المتغير ورؤية قيمته أو تغييرها مباشرة من تلك النافذة و بنفس الطريقة يمكنك أيضا إضافة Watch لأحد العناصر المركبة ورؤية أو تغيير قيمة إحدى خصائصه

لإزالة متغير من نافذة Watch فقط انقر بزر الفأرة اليميني عليه في تلك النافذة واختر Delete Watch

كما يمكنك كتابة اسم المتغير مباشرة في نافذة watch لمراقبته

في حالة وجود كمية بيانات كبيرة أو بنية بيانات معقدة داخل المتغير كبيانات XML مثلا يمكنك ملاحظة أيقونة مكبرة بجانب تلك القيمة حيث يمكنك إما الضغط على المكبرة مباشرة لعرض البيانات أو النقر على السهم الصغير بجانبها لاختيار طريقة عرض تلك البيانات من القائمة حيث يمكنك اختيار Xml Visualizer مثلا في حالة بيانات من نوع XML

يمكنك استخدام نقاط التعقب Trace Points ليقوم المنقح بتنفيذ عمل معين عند وصوله لهذه النقطة دون إيقاف تنفيذ البرنامج أو مع إيقاف التنفيذ

لوضع نقطة تعقب Tracepoint انقر بزر الفارة اليميني على سطر الكود ثم من القائمة الفرعية Breakpoint اختر Insert Tracepoint وهذا يؤدي إلى ظهور مربع حوار When Breakpoint Is Hit الذي يمكنك من تحديد ماذا تريد أن يفعل عندما يصل التنفيذ لذلك السطر حيث يوفر لك إمكانية طباعة رسالة أو تنفيذ ماكرو بالإضافة إلى خيار لاستمرار التنفيذ أو إيقافه عند ذلك السطر كما يمكنك استخدام تعابير معينة لإظهار قيم خاصة في سطر الرسالة مثل $TICK لإظهار استخدام المعالج أو $TNAME لإظهار اسم مسار التنفيذ الحالي Current Thread Name وعند ضبطها ستلاحظ ظهور معين أحمر بجانب السطر دلالة على Trace Point عوضا عن الدائرة الحمراء التي تشير لـ Break Point وستظهر الرسائل المتعلقة بـ Trace Point في نافذة Output

الاستثناءات Exceptions اصطياد الأخطاء ومعالجتها

التقاط استثناء معين

لالتقاط استثناء نستعمل بلوك Catch ... Try بشكل عام عندما ينفذ البرنامج عملية معينة قد تولد استثناء فلعمل ذلك نقوم بوضع تلك الشيفرة البرمجية بين عبارتي Try و Catch و بعد العبارة Catch نستكشف الاستثناءات الحاصلة

```
Try
      C = A + B
Catch Ex as OverflowException
```

ويتيح الجزء Catch للبرنامج اكتشاف استثناء معين والرد عليه فمثلا يمكننا التقاط استثناء القسمة على صفرDivided By Zero

```
Try
      C = A Mod B
      TextBox3.Text = C.ToString()
Catch Ex as DividedByZeroException
      MsgBox("Devided By Zero.")
      TextBox3.Text = "Infinity"
End Try
```

وبنفس الطريقة يمكننا استكشاف استثناء فيضان Overflow Exception

```
Dim A, B, C As Integer

Try
      A= TextBox1.text
      B = TextBox2.Text

      C = A + B
Catch Ex as OverflowException
      MsgBox("Overflow.")
      TextBox3.Text = "Infinity"
End Try
```

وحتى أيضا يمكننا استخدامه للكشف عن اسم ملف غير صالح

```
Dim Fname As New String = "D:\Some Folder\FileName.ext"
Dim Sfl as new StreamReader
Try
      Sfl = New StreamReader(Fname)
      TextBox1.Text = Sfl.ReadToEnd()
      Sfl.Close
Catch Ex As FileNotFoundException
      MsgBox("File Not Found")
End Try
```

فحص عدة استثناءات

عندما يمكن أن تؤدي العملية التي تقوم بتنفيذها إلى عدة استثناءات مختلفة يمكنك تحديد سلسلة من الجمل Catch لمعالجة تلك الاستثناءات

```
Dim FileDB As New OpenFileDialog()
```

```
FileDB.Filter = "All files | *.* | Text files | *.txt"

FileDB.FilterIndex = 2
FileDB.InitialDirectory = "C:\Temp"
FileDB.AddExtension = True
FileDB.DefaultExt = "txt"

' Prevent dialog box from validating file
FileDB.CheckFileExists = False
FileDB.CheckPathExists = False

If (FileDB.ShowDialog() = DialogResult.OK) Then
      Dim SourceFile As StreamReader

      Try
            SourceFile = New StreamReader(FileDB.FileName)

            TextBox1.Text = SourceFile.ReadToEnd()
                  SourceFile.Close()
      Catch Except As DirectoryNotFoundException
            MsgBox("Error: " & Except.Message)
      Catch Except As FileNotFoundException
            MsgBox("Error: " & Except.Message)
      Catch Except As Exception
            MsgBox("Error: " & Except.Message)
      End Try
Else
      MsgBox("User selected Cancel")
End If
```

معالجة الاستثناءات باستعمال بلوك Catch عام

عندما ينفذ كائن عملية نيابة عن البرنامج فقد يولد نطاقا واسعا من الاستثناءات بناء على سير تنفيذ البرنامج وقد لا تهمك ما هي هذه الاستثناءات بقدر ما يهمك أنه قد حصل هناك استثناء ما ولمعالجة الاستثناءات بغض النظر عن نوعها فإننا لا نحدد استثناء معينا بل نستخدم

```
Catch Ex as Exception
```

مثال

```
Try

      ..... Some Code Here

Catch Ex As Exception
      MsgBox("Error: " & Ex.Message)
End Try
```

إجراء التنظيف بعد حدوث استثناء

عند استعمالك لبلوك Catch ... Try للرد على سلسلة من الاستثناءات ستنفذ عادة عمليات تخص كل استثناء ضمن بلوك Catch المناسب وبناء على الأمور التي يقوم بها برنامجك عليك القيام بعم لهات معينة بعد حدوث استثناء وذلك بغض النظر عن نوع الاستثناء ولهذا الغرض نستخدم عبارة Finally في نهاية بلوك Catch ... Try تحدد الجمل التي نريد تنفيذها بغض النظر عن نوع الاستثناء مع ملاحظة أن العبارات الموجودة ضمن بلوك Finally سيتم تنفيذها دوما بغض النظر عن حدوث استثناء أو لا

```
Dim FileDB As New OpenFileDialog()

FileDB.Filter = "All files | *.* | Text files | *.txt"
```

```
FileDB.FilterIndex = 2
FileDB.InitialDirectory = "C:\Temp"
FileDB.AddExtension = True
FileDB.DefaultExt = "txt"

' Prevent dialog box from validating file
FileDB.CheckFileExists = False
FileDB.CheckPathExists = False

If (FileDB.ShowDialog() = DialogResult.OK) Then
      Dim SourceFile As StreamReader

      Try
            SourceFile = New StreamReader(FileDB.FileName)
      Catch Except As Exception
            MsgBox("Error: " & Except.Message)
      End Try

      If (Not SourceFile Is Nothing) Then
            Try
                  TextBox1.Text = SourceFile.ReadToEnd()
            Catch Except As Exception
                  MsgBox("Error: " & Except.Message)
            Finally
                  MsgBox("In finally statements")
                  SourceFile.Close()
            End Try
      End If
Else
      MsgBox("User selected Cancel")
End If
```

وفي بعض الحالات قد تكون هناك أوقات لا تريد استكمال تنفيذ البلوك Catch ... Try عندها تستخدم العبارة Exit Try للخروج من البلوك حيث سينفذ بعدها أول سطر كود يلي End Try

```
Try

      .... Some Code

      If SomeCondition Then Exit Try

Catch Ex as Exception

  Exit Try

  .... Rest of Try Block
```

إطلاق استثناءاتك الخاصة

هناك أوقات تحتاج فيها لتكوين استثناء خاص بك عندها ستحتاج بكل بساطة لإنشاء فئة Class ترث الفئة - Exception فعلى سبيل المثال يمكننا توليد استثناء باسم InvalidEMailException كما يلي

```
Public Class InvalidEMailException
      Inherits System.Exception
```

```
	Sub New(ByVal Message As String)
		MyBase.New(Message)
	End Sub
End Class
```

و بالطبع يمكنك إنشاء طرق وخصائص في هذه الفئة حسب احتياجاتك   كأي فئة أخرى وفي مثالنا المبسط هنا أنشأنا مشيد الفئة فقط وبعد إنشاء فئة الاستثناء الخاصة بنا يمكننا توليد الاستثناء باستخدام العبارة Throw

```
Throw New InvalidEMailException("Envalid Email Please Correct")
```

وفيما يلي مثال آخر

```
Public Class MyException
	Inherits System.Exception

	Sub New(ByVal Message As String)
		MyBase.New(Message)
	End Sub
End Class

Public Class Form1
	Inherits System.Windows.Forms.Form

	.....

	Private Sub Button1_Click(ByVal sender As Object, _
			ByVal e As System.EventArgs) Handles Button1.Click

		Try
			MsgBox("About to generate custom exception")
			Throw (New MyException("** Custom Message **"))
		Catch Ex As MyException
			MsgBox("Custom Exception thrown " & Ex.Message)
		End Try

	End Sub

	......

End Class
```

ترشيح الأخطاء في قسم CATCH في بلوك TRY عند اصطياد الأخطاء

يوفر لنا قسم Catch في بلوك Try أكثر من خيار لترشيح الأخطاء وإحدى هذه الطرق هي بتحديد نوع الخطأ المراد اصطياده وهنا يجب عليك البدء بالنوع الأكثر تحديدا منتهيا بالنوع الأكثر عمومية بما أن قسم Catch يتم تنفيذه بترتيب كتابته كما يمكن استخدام When في قسم Catch لتحديد أكثر دقة مثل تحديد رقم خطأ معين حيث يمكنك دمج هذه الأساليب للحصول على الطريقة المناسبة لبرنامجك

مثال:

```
Try
	' خطأ يسبب ان المحتمل كودك
Catch ex As System.IO.IOException
	' كود للتفاعل مع الخطأ IOException
Catch ex As System.NullReferenceException
	' كود للتفاعل مع الخطأ NullReferenceException
Catch ex As Exception When Err.Number = 5 Or Err.Number = 8
	' كود للتفاعل عندما يكون رقم الخطأ 5 أو 8
Catch ex As Exception
	' كود للتفاعل مع أي خطأ آخر
End Try
```

فيما يلي بعض فئات الأخطاء ووصف سريع لكل منها

الفئة	الوصف
AmbiguousMatchException	لم يستطع البرنامج تحديد أي نسخة من الدالة يستخدم
ApplicationException	هذه هي الفئة الأب لجميع فئات الأخطاء الغير قاتلة. فعندما تبني فئة استثناء خاصة بك يحب أن تكون موروثة من هذه الفئة
ArgumentException	القيمة الممررة غير صحيحة
ArgumentNullException	القيمة الممررة لا يمكن أن تكون لاشئ ومع ذلك قيمتها لاشئ
ArgumentOutOfRangeException	القيمة الممررة خارج المجال المقبول
ArithmeticException	خطأ إسناد أو تحويل في عملية حسابية
ArrayTypeMismatchException	البرنامج يحاول القيام بإدخال عنصر من نوع خاطئ في المصفوفة
ConfigurationException	قيمة الإعداد غير صحيحة
ConstraintException	عملية البيانات تسبب خطأ في القيود على قاعدة البيانات
DataException	الفئة الأب لجميع فئات الأخطاء المتعلقة بـ ADO .NET
DirectoryNotFoundException	المجلد المطلوب غير موجود
DivideByZeroException	خطأ القسمة على صفر
DuplicateNameException	عملية ADO .Net واجهت اسما مكررا. مثل أنك تحاول إنشاء جدول مع أنه يوجد جدول موجود سابقا ويملك نفس الاسم
EvaluateException	يحدث عندما لا يستطيع البرنامج تقييم قيمة التعبير الموجود في عمود قاعدة البيانات
FieldAccessException	البرنامج يحاول الوصول إلى خاصية للفئة بطريقة غير صحيحة
FormatException	تنسيق القيمة الممررة غير صحيح
IndexOutofRangeException	البرنامج يحاول الوصول إلى عنصر يقع خارج حدود المصفوفة أو أي عنصر احتواء آخر
InvalidCastException	البرنامج يحاول القيام بتحويل نوع غير صحيح
InvalidOperationException	العملية المطلوبة حاليا غير مسموح بها
IOException	الفئة الأب لجميع فئات أخطاء الدخل/الخرج. خطأ دخل/خرج عام
EndOfStreamException	وصل الـ Stream إلى نهايته
FileLoadException	خطأ أثناء تحميل الملف
FileNotFoundException	لا يمكن إيجاد الملف المطلوب
InternalBufferOverflowException	حدوث فيضان في الـ Buffer الداخلي
MemberAccessException	البرنامج يحاول الوصول إلى عنصر في فئة بطريقة غير صحيحة
MethodAccessException	البرنامج يحاول الوصول إلى الطريقة بطريقة غير صحيحة
MissingFieldException	البرنامج يحاول الوصول إلى خاصية غير موجودة في الفئة
MissingMemberException	البرنامج يحاول الوصول إلى عنصر غير موجود في الفئة
MissingMethodException	البرنامج يحاول الوصول إلى طريقة غير موجودة في الفئة
NotImplementedException	العملية المطلوبة غير معرفة
NotSupportedException	الخاصية المطلوبة غير مدعومة
NullReferenceException	البرنامج يحاول استخدام مرجع إلى غرض Object قيمته لاشئ
OutOfMemoryException	لاتوجد ذاكرة كافية فمثلا إن كان المستخدم يحاول توليد مجموعة بيانات ضخمة يمكنك التنبؤ بحجم الذاكرة التي سيحتاجها البرنامج واختبار إن كانت متوفرة فتقوم برمي هذا الاستثناء إن لم تكن كافية
OverflowException	خطأ فيضان في عملية حسابية
RankException	الإجراء يحاول استخدام مصفوفة تملك عددا خاطئا من الأبعاد
ReadOnlyException	البرنامج يحاول تعديل بيانات للقراءة فقط
ResourceException	المصدر المطلوب مفقود
SyntaxErrorException	خطأ في الصيغة عند إسناد قيمة لخاصية
UnauthorizedAccessException	النظام يمنع الوصول بسبب عدم كفاية الصلاحيات

التقاط الاستثناءات الغير معالجة في التطبيق

في Application Events يوجد الحدث UnhandledException الذي يستقبل جميع الاستثناءات الغير معالجة في التطبيق مع ملاحظة أنه عندما يصل الاستثناء لهذا الحدث فإن التطبيق سيتم إنهاؤه و لا يعود مسار التنفيذ لداخل التطبيق